Práctica de búsqueda local curso 2017-2018 Q1

Documentación de la realización de la práctica y los experimentos

Sebastián Sánchez Menéndez

Daniel Martínez Bordes

Ferran Martínez Felipe

Sophia Lichtenberg

Representación del estado

Para solucionar el problema de la planificación de rutas de abastecimiento de las gasolineras mediante algoritmos de búsqueda local, el primer paso que debemos realizar es la definición de la representación del estado que será utilizada. Esto implica analizar los elementos que influyen en la solución del problema. Contamos con un conjunto de camiones, que tienen que atender ciertos pedidos pertenecientes a gasolineras, y debemos asignar pedidos a los camiones, cumpliendo unos ciertos criterios, de forma que se optimice el beneficio obtenido. Es decir, los elementos básicos que intervienen en el problema son los camiones, y los pedidos.

Un camión se puede identificar mediante un entero: su posición en la clase CentrosDistribucion. Sin embargo, un pedido necesita dos enteros para ser identificado: su posición en la clase Gasolineras, y dentro de esa instancia de Gasolinera, su posición en el vector de peticiones.

Por lo tanto, hemos decidido implementar una clase Order que representa una petición, con sus correspondientes atributos.

Como posible representaciones del estado proponemos las siguientes:

* + Un diccionario, de la forma <Order, int>, de forma que a cada orden le corresponde un camión. Las ordenes no asignadas quedarían representadas por un value **null.**
  + Un vector, donde a cada posición le corresponde otro vector de tipo Order. Cada fila del vector representa un camión, y el vector que contiene son las Ordenes que tiene asignadas. Dado un número de camiones C, el vector tendría C+1 vectores de órdenes, donde el último vector son las órdenes no asignadas. Como los camiones pueden realizar dos pedidos en un solo viaje, decidimos implementar una clase Trip que agrupa dos pedidos.

Dadas estas dos representaciones, vamos a pensar en las implicaciones temporales que tiene cada una de ellas. Al realizar cualquier operación posible sobre los peticiones y los camiones (asignar una petición no asignada a aun camión, intercambiar una petición entre camiones, o mover una petición de un camión a otro), siempre hemos de comprobar que no nos salimos del espacio de soluciones, comprobando las restricciones sobre el número de viajes y la distancia recorrida. En la segunda representación, dado que podemos acceder directamente a todas las peticiones asignadas a un camión, podemos hacer este cálculo de forma constante recorriendo sus peticiones. Sin embargo, en la representación en forma de diccionario deberíamos recorrer todas las peticiones para saber cuales pertenecen al camión que alteramos, por lo que tendríamos una complejidad temporal T= O(nº peticiones). La aplicación de los operadores es constante en cualquier caso, ya que el acceso al pedido es constante.

Otro problema de la representación en forma de diccionario es que mantiene información sobre a qué camión está asignada cada petición, pero no en qué orden se realizan esas peticiones. Dado que en un viaje se pueden hacer dos peticiones, la forma de agruparlas es importante, porque afecta a la distancia recorrida y por tanto al cálculo del beneficio.

Por tanto, elegimos la representación del estado en forma de vector de vectores de Trip, que se implementa en Java con la clase ArrayList < ArrayList < Trip > >. Llamaremos a esta estructura trips.

Para acceder a un pedido se hará mediante tres variables, i, j, y k. La variable i representará el camión al que accedemos, la variable j representa el número de viaje y la variable k representa el número de pedido dentro de ese viaje (como mucho hay dos pedidos por viaje).

Generación de la solución inicial

Como posibles soluciones iniciales a estudiar tenemos 4 variantes:

* Solución vacía: permite que el algoritmo trabaje más con operadores de adición. Se trata de introducir todas las peticiones generadas en el camión fantasma (trips[C]).
* Solución llena, aleatoria: Se trata de introducir cada una de las peticiones en un camión aleatorio.
* Solución llena, ordenada: Se trata de introducir las peticiones, en orden, dentro de los distintos camiones hasta llenarlos (según los criterios descritos) y pasar al siguiente. Las peticiones restantes pasan al camión fantasma.
* Solución llena, greedy: Se trata de, para cada camión, introducir las órdenes no asignadas más cercanas. Tendremos que ordenar las órdenes no asignadas para cada camión que empezamos a llenar.

Generadora de sucesores

Experimentación

Experimento nº 1

En este experimento queremos comparar los distintos conjuntos de operadores propuestos en función de la calidad de la solución obtenida con cada uno de ellos. Definiremos que una solución es mejor que otra si la relación entre la puntuación obtenida y el tiempo de ejecución del algoritmo es mayor. Es decir, el parámetro a estudiar es P/T.

Nuestra hipótesis nula es que todos los conjuntos de operadores se comportan de igual manera.

Para realizar el experimento, partiremos de una solución vacía en todos los casos.

La realización del experimento será la siguiente: se escogerán 20 semillas de forma aleatoria. Para cada una de estas semillas, se hará una única ejecución con cada conjunto de operadores. Después, haremos medias del parámetro P/T para cada conjunto de operadores y los compararemos.